UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/784,026	02/20/2004	Laurent D. Hasson	YOR920030638US1	1688

34663 7590 06/19/2008

MICHAEL J. BUCHENHORNER
8540 S.W. 83 STREET
MIAMI, FL 33143

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

NOTIFICATION DATE	DELIVERY MODE
06/19/2008	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

MICHAEL@BUCHENHORNER.COM
ANA@BUCHENHORNER.COM

Office Action Summary	Application No.	Applicant(s)	
	10/784,026	HASSON ET AL.	
	Examiner	Art Unit	
	Tuan A. Vu	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *10 April 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1 and 3-14* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1 and 3-14* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 4/10/08.

As indicated in Applicant's response, claims 1, 3-6, 11, 13-14 have been amended. Claims 1, 3-14 are pending in the office action.

Claim Objections

2. Claims 1, 13, 14 are objected to because of the following informalities: the language recited as 'maintaining rules from the instance data that enforce Java modeling constraints not found in JavaScript' and 'invoking the converters at runtime … Javascript code for projecting onto the browser client" contain some impropriety in language usage that require correction. Example is taken in analyzing claim 1. First, the 'enforce' verb should be corrected to indicate that the subject is 'instance data', or re-arranged to convey that 'rules' are subject to this verb. Second, 'rules from the instance data' does not convey that rules are coming from a model but rather from mere *instance data*, which is not commensurate with the step for creating converters, step c) which is making use of 'instance data'. Third, the term 'runtime' in step f) as to invoke converters cannot be commensurate with 'compiled recursive descent parser' representing the converters, a parser which pertains to a translation process with rule enforcing. Runtime (without a clear environmental setting) is usually interpreted as for executing compiled code after a step of code translation, whereas 'parser' used to enforce rules cannot be same as 'runtime' which appears to be invoking this parser. Some language readjustment regarding invoking the converters is required.

3. Further, the language recited as 'JavaScript code *for recreating an instance* from the classes as JavaScript objects for display on a browser client of the web server' entails code for

recreating an instance from classes as JS object for display; but as phrased, this code is operable for recreating Java objects in form or more JS object on some browser, such that the JS object represents a recreated instance of a Java class. This recreated Java class cannot be found as evidence in the Disclosure. The language for 'recreating' should be corrected to merely convey a re-conversion process, and the wordiness of 'recreating an instance from the classes as JavaScript objects for display on a browser client of the web server' would necessitate more precise syntax. The Specifications does not separate JS code and JS 'object for display' such that JS object for display is recreated as from running JS code (see Specs pg. 14, 17, 25), nor does the Specifications remotely mentions about 'recreating instance from the classes' anywhere that concerns a converter.

4. Claims 13 and 14 are objected to because of the same improprieties in syntactic usage. Appropriate correction is required.

The 'parser' will be treated as for enforcing rules of the Java model in converting Java objects into JavaSript; whereas the 'runtime' is treated as a compilation step to generate JavaScript code. Moreover, the 'recreating' will be treated as converting Java class into JS objects represented as JS code usable at a browser end.

Claim Rejections - 35 USC § 101

5. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

6. Claim 13 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

The Federal Circuit has recently applied the practical application test in determining whether the claimed subject matter is statutory under 35 U.S.C. § 101. The practical application test requires that a " useful, concrete, and tangible result" be accomplished. An "abstract idea" when practically applied is eligible for a patent. As a consequence, an invention, which is eligible for patenting under 35 U.S.C. § 101, is in the "useful arts" when it is a machine, manufacture, process or composition of matter, which produces a concrete, tangible, and useful result. The test for practical application is thus to determine whether the claimed invention produces a "useful, concrete and tangible result".

The current focus of the Patent Office in regard to statutory inventions under 35 U.S.C. § 101 for method claims and claims that recite a judicial exception (software) is that the claimed invention recite a practical application. Practical application can be provided by a physical transformation or a useful, concrete and tangible result. The following link on the World Wide Web is for the United States Patent And Trademark Office (USPTO) policy on 35 U.S.C. §101. <http://www.uspto.gov/web/offices/pac/dapp/opla/preognotice/guidelines101_20051026.pdf>

Specifically, claim 13 discloses a *system* comprising a *web server* (i.e. Specifications: element 108 in Fig. 1) *comprising Java objects*, a browser, a NW interface between said browser and server; a program for converting with instructions for *identifying, determining, introspecting, creating*, and *generating*. The system as claimed and understood based on the *web server*--being viewed as Java objects, a browser application and underlying browser interface; hence amounts to a mere listing of software instructions, application interface or objects. Mere reciting of software entities (e.g. object, browser, browser interface) cannot be construed as a method, a apparatus, a article of manufacture or composition of matter, i.e. not one of the four categories permitted for a 101 type of subject matter. According to the § 101 Guidelines (see Annex IV(a), pg 53-54) for it falls under the subject matter identified as 'Functional Descriptive Material' and cannot be considered sufficiently supported with hardware in a tangible functional form in order to realize the software into real-world tangible results. The claim as a whole is rejected for leading to a non-statutory subject matter.

Claim Rejections - 35 USC § 112

7. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

8. Claims 1, 3-14 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Claim 1 recites (i) 'artifact mirroring the Java-based object hierarchy' and (ii) 'converters together form a compiled recursive descent parser, synching the object hierarchies by maintaining rules from the instance data that enforce Java modeling constraints not found in JavaScript". The Disclosure makes no mention whatsoever about 'hierarchy' of objects anywhere in the context of 'artifact mirroring' (step d), invoking or creating a parser (step e), or 'maintaining rules' (step e); whereas the correspondence between 'synching… hierarchies' in the context of rule-maintaining from the *instance data that enforce* 'Java modeling constraints not found in JavaScript' is nowhere described in the Disclosure. It is deemed that the inventor has no possession of the above limitation at the time of the Invention. Dependent claims 3-12 are equally rejected for failing to remedy to this lack of description in the Disclosure.

Claims 13, 14 are rejected for recited the same unsupported language.

9. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

10. Claims 1, 3-14 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

11. Specifically, claim 1 recites 'artifact mirroring the Java-based object *hierarchy*' (line 7); there is no sufficient antecedent basis for the term 'hierarchy' in the claim, rendering its introduction indefinite; and this term will be treated as mere Java-based object resulting from introspection.

12. Claim 1 recites 'converter ... for receiving the instance data and generating Javascript code for recreating *an instance from the classes* as JavaScript-based objects for display on a browser client of the server' (lines 9-11). **First**, it is not clear how the language 'an instance from the classes' can be referenced back to the previously recited elements of the claim. An 'instance' as recited is indefinite because it is not certain if this, inter alia, relate to 'instance data' from step b) or a generic instantiated object of the Java-based classes. **Second**, 'instance from the classes' does not make it clear whether *classes* refer to classes prior to the introspecting step or after step c) or one class for a corresponding converter in step e). Broad interpretation or no patentable weight will be applied for this 'an instance from the classes'. **Third**, the language recited as 'Java-script based objects for display on a *browser client* of the server' is indefinite. Common accepted lexicographic definition of the terms 'client', 'server' and 'browser' or even 'client's browser' cannot be extended such to condone terminology such as 'browser client of the server', unless the instant Disclosure provides explicit redefinition of this phrase, which is not the case. The phrase will be treated as 'browser' targeted to receive this code. **Fourth**, claim 1 recites 'browser client' in step f) without definition of this term in the Disclosure; i.e. the weight given

to this idiomatic terminology will be as set forth above. **Fifth**, claim 1 recites 'synching the object hierarchies' by maintaining rules. There is no sufficient antecedent basis for the 'object hierarchies' based on the language in steps a) to e). The 'hierarchies' term will be given no patentable weight beyond Java model data.

Claims 3-12 for not remedying to the above improprieties will be rejected likewise.

13. Claim 13 for reciting the same improprieties in antecedent basis and idiomatic terminology as laid out in claim 1, is also rejected for the same reasons.

14. Claim 14 is rejected for reciting the same indefinite language or terminology as in claim 1, therefore equally rejected.

Claim Rejections - 35 USC § 103

15. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

16. Claims 1, 3-14 are rejected under 35 U.S.C. 103(a) as being unpatentable over Jim Conallen , "Modeling Web Application Architectures with UML", October 1999, ACM, Vol 42, No 10, pp. 63-70 (hereinafter Conallen), in view of Chung et al, "Modeling Web Applications Using Java and UML Related Technologies', Proceedings of the 36th HICSS'03, IEEE 2002 pp. 1-10 (hereinafter Chung), and further in view of Kuznetzov, USPN: 6,772,413 (hereinafter Kuznetzov).

As per claim 1, Conallen discloses a server method for converting objects of a first type into objects of a second type, the method comprising:

identifying one or more Java-based object classes (e.g. *Web pages ... class abstraction of logical behavior* – pg. 68, bottom L col. to top R col.; Fig. 3, pg. 68) ;

determining instance data based on the Java-based object classes (e.g. Fig. 3; UML … custom icon – pg. 68, bottom L - Note: runtime nature of Web pages being passed from client to server being reconstructed at server page reads on OO instance data being introspected at server to map a UML construct, i.e. such instance leading to a UML icon -- see *variable number of instances* – L top col., pg. 70);

introspecting each Java-based object class and automatically creating an artifact representing a software model (e.g. pg. 68: bottom L col. to top R col. - Note: runtime data of Web pages being from client to server reads on derived *artifacts* -- details inside each UML iconic structures -- based on OO instances from Web pages introspected at server to derive a corresponding UML construct; Fig. 3), the model describing instance data, the artifact mirroring Java-based object hierarchy (Note: model and artifact treated as Java-based object mirroring Java classes); and

generating one or more converters (e.g. <form> - R col, bottom, pg. 69; HTML input tags - pg. 70, L top para) for each Java-based object class, each converter being based on the artifact and configured for receiving the instance data.

Conallen does not explicitly disclose generating JavaScript code for recreating the instance from the classes as objects of the second type, for display on a browser. However, the client/server system by Conallen clearly discloses Javascript based client side communicating with server UML-based framework to create Web applications (e.g. "Java Script item" - Fig. 4, pg. 69) and Web forms or frame navigation supporting client pages (pg. 70, L top paras;

navigation of Web pages - pg. 64, L col; Fig. 5, pg. 69;) to be utilized in relation with server in business transactions and a converter to output formatted stream to address requests the client browser navigation scheme or underlying HTML or Java script execution (see HTML formatted stream … sent back - pg. 66, middle R col.; Fig. 4, pg. 69). The need to provide Javascript-based form of HTML/Web executable constructs to support the client for effectuating a business transaction or to render, at the Jvscript-based client side the data sent back from the server is suggested from the above. The type of components that can be modeled in UML (see Chung: Fig. 6, pg. 9) and deployed at workstations as scripts (see Chung: sec 4, 5.1, pg. 3) are taught in Chung; and accordingly, Chung discloses that JavaScript component can be modeled with class (see Table 4, pg 6); that client-side script programming being Javascript (see pg. 3, bottom L col) and Javascript to validate data inside a Web HTML frame. Based on possibility that modeling can be used to implement JavaScript and the role played by JavaScript on client side application as by Chung, and in view of the need to visualize the content of runtime transaction-related pages as see in Conallen(see Fig. 5, pg. 69), it would have been obvious for one skill in the art at the time the invention was made to implement the conversion in Conallen's server developing tool so that the returned formatted stream as by Conallen be effectuated to contain the server-side generated Javascript constructs (i.e. OO class of second type). That is, one would be motivated to do so because the generated JVscript code would support the deployment of business web-based applications navigation as in Conallen's client side, the support as conveyed in Chung's establishing UML/Javascript relationship along with necessary Javascript dependency within deployment of application content (e.g. Email application, see Chung Fig. 6) at the requesting client as endeavored in Conallen's paradigm.

Nor does Conallen explicitly disclose 'converters form a compiled recursive descent parser, synching object hierarchies by maintaining rules from the instance data that enforce Java modeling constraints not found in JavaScript'; and 'invoking the converters at runtime to directly generate JavaScript code for projecting onto the browser client'. Based on the rationale using converter to recreate JavaScript objects (Note: rules in Java based on UML would read on *constraints not found in JavaScript*) on a browser as set forth above, the limitation as to invoking of runtime converters would have been obvious, accordingly. Enforcing rules derived from UML class objects and underlying definitions in UML or Rational Rose is deemed integral to Conallen's applying of UML in creating artifact or instance data as set forth above. And parsing model constructs requires a internal tree traversal was a well-known concept at the time the invention was made. Commensurate with the suggestion by Conallen that XSLT in conjunction with use of HTML type of document like XML in the rendering of content via use browser script or page content various for of control (see Conallen: pg. 66, annotation R bottom), Kuznetsov discloses using a XSL converter in conjunction with parsing of XML format, and applying of a recursive type of parser provided by W3C consortium (e.g. col. 2 lines 40-62) to go down elements of the stylesheet tree nodes (see Kuznetsov: Fig 8A, Fig. 9; col. 5, lines 42-57) to translate into target code. It would have been obvious for one skill in the art at the time the invention was made to implement the script destined for Conallen's browser so that the elements of the XML type document are rendered by script with use of a XSLT parser with recursive descent type in order to traverse each node elements representing the markup hierarchy for being rendered at the target browser, thereby all the elements of this tree (as set forth above) are

addressed recursively based on the well-known methodology provided by W3 Consortium to validate each and every elements of a markup language.

As per claim 3, Conallen discloses wherein the converters are invoked at runtime to generate JavaScript code (in light of the combination with Chung) required to recreate instance data from the JavaBeans into JS objects in the browser; determining, for each property in a class, whether the property is an attribute (see class attribute – pg. 68, bottom L col.). But Conallen does not disclose that applets in a client Web Application (see Fig. 2, pg. 68) comes from server building based on class objects from Javabeans. But Chung teaches JavaBeans for server-side applications and modeling a EmailBean via introspecting the class attributes (see bottom L col. pg. 3 to top R col. pg. 4; *attributes JavaBean* – sec 5.3, pg. 5, R col). Based on the above, it would have been obvious for one skill in the art at the time the invention was made to implement the client side applets by Conallen from forming the code in the server UML framework based on beans as taught above for which the analysis of class attributes as in Chung's reusable package (e.g. *what packages are used ... easily by using class notation in UML* - sec 5.3, pg. 4) prove (i) an efficient way of reusing Java code and (ii) easy Java Class mapping with UML notation, thereby enabling proper support of the client applet executed at Conallen's browser runtime, as mentioned above.

As per claim 5, Conallen does not explicitly disclose (wherein invoking comprises) outputting a JavaScript code for the value of the attribute. But based on the attribute of Javascript (see JavaScript Participant, Fig. 2-3,pg. 68; JavaScript item, Fig. 4, pg. 69) from Conallen, and the rationale in claim 1, this outputting in Jvscript code including value of the attribute would also have been obvious in view of the above obvious reasons.

As per claim 6, Conallen does not explicitly disclose creating a buffer to hold an exported object value or values if the property is an array. At the time the invention was made, it was well-known concept that *Javascript* does not support a predefined type as with strongly typed languages like C++ or Java, and that an array in Javascript can have no fixed bound as it by itself grows dynamically upon execution unless its bounds have to be customized by user's code; and this is suggested in Conallen (see pg 70: type-less language). Based on the communication paradigm to provide stream of formatted data in order to support the client-side rendering of Web pages via server-side generating of Javascript code as rationalized in claim 1 and Conallen's analysis of bean-class attributes as set forth above, it would have been obvious for one skill in the art at the time the invention was made so that Conallen's exporting of attributes for a Javascript code to the client Web pages for rendering would include provision for Conallen's attributes enumerated or listed in an array (see Fig. 2-3, pg. 68), in view of the above dynamic expanding of array size; that is, provision in form of creating buffer to hold values of the exported array because this would accommodate to the type bound deficiency of Javascript with the runtime requirements of a given business transaction effectuated using Javascript on the client pages.

As per claim 7, the limitation creating an array to hold object IDs for the referenced objects for each object in the array for holding all the attributes of Java class implementing a transaction functionality in Conallen in form of Javascript code would falls under the obviousness rationale as set forth in claims 1 and 6.

As per claim 8, Conallen does not explicitly disclose computing a signature of the object if the object is in a map. But the analysis of UML constructs entail a plurality of map

instance between a UML notation representing a client functionality and a corresponding server side UML equivalent (see Fig. 2-3, pg. 68); and since the providing of attributed in Java being effectuated to support export of object class to implement the Javascript code for the client applications, the class being viewed in Conallen's mapping would necessitate a constructor for the Javabeans as set forth in claim 3. Hence the limitation as creating a signature for a beans (or its constructor) would have been an obvious limitation based on the code generating of Java constructs as set forth in claim 3.

As per claims 9-10, Conallen is not explicitly disclosing the steps of calling an object; generating an export ID; outputting the JavaScript code to declare the object called as the converter for the object; and adding the ID to the array. But these are obvious steps for standard construction of a class beans in view of claim 3, the array of claim 7, and the plurality of attributes identified (see claims 6-7) when creating the buffer and array for exporting the Javascript-based object (using a export ID object map as in claim 8) including populating the array with attributes ID for such export object destined for the client Web page execution environment as set forth in claim 1; that is Conallen discloses (in view of the above obvious constructor signature and array ID) outputting a JavaScript array to hold the exported IDs.

As per claim 11, the limitation as to determine an UML object signature and a map therefor has been obvious in light the rationale in claim 8; and the limitation as to check whether the object already in the Map get its export ID is deemed a obvious step for validating a Java construct to enable all the attributes to be included in the Javascript implemented with a Map for export to the client Javascript-based runtime, lest the execution of said Javascript would incur for example, null pointer exception.

As per claim 12, Conallen (in view of claim 9) discloses calling the converter for the object based on the rationale in claims 1 and 3; and outputting the buffer for the exported object values based on the rationale of claim 6.

As per claim 13, Conallen discloses an information processing system comprising:

a web server comprising one or more Java objects (Fig. 3, pg. 68); a web browser that display JavaScript objects (*On the client side... HTML browsers* -pg. 69, L bottom – Note: client browser reads on operable with and rendering with JavaSript code or objects); a network interface between server and web browser(Web server, HTTP browser - pg. 63); and

a web server program for converting the Java objects to JavaScript objects (Fig. 2-3, 4, pg. 68-69); wherein the program comprises instructions for:

a) identifying one or more Java-based object classes;

b) determining instance data based on the Java-based object classes;

c) introspecting each he Java-based object class;

d) automatically creating an artifact representing a software model (describing the instance data … Java-based object hierarchy); and

e) generating one or more converters for each he Java-based object class, each converter being based on the artifact and configured for receiving the instance data.

Conallen does not explicitly disclose generating JavaScript code for recreating the instance from the classes as objects of the second type, for display on a browser.

Nor does Conallen explicitly disclose 'converters form a compiled recursive descent parser, synching object hierarchies by maintaining rules from the instance data that enforce Java

modeling constraints not found in JavaScript'; and 'invoking the converters at runtime to directly generate JavaScript code for projecting onto the browser client'.

But these limitations have been rendered obvious by virtue of the corresponding rationale as set forth in claim 1.

As per claim 14, Conallen (in view of Chung) disclose a computer readable medium comprising program instructions for:

a) identifying one or more Java-based object classes;

b) determining instance data based on the Java-based object classes;

c) introspecting each Java-based object class;

d) automatically creating an artifact representing a software model (describing .... Java-based object hierarchy);

e) generating one or more converters .... , each converter being based on the artifact and configured for receiving the instance data;

and in view of Chung,

generating JavaScript code for recreating the instance from the classes as objects of a second type, for display on a client browser (refer to the combination with Chung in claim 1); and

wherein the converters form a compiled recursive descent parser, synching object hierarchies by maintaining rules from the instance data that enforce Java modeling constraints not found in JavaScript' (refer to claim 1 for corresponding rejection); and

f) invoking the converters at runtime to directly generate JavaScript code for projecting onto the browser client' (refer to claim 1)

Response to Arguments

17. Applicant's arguments filed 4/10/08 have been fully considered but they are not persuasive. Following are the Examiner's observation in regard thereto.

35 USC § 101 Rejection:

(A) Applicants have submitted that (Appl. Rmrks pg. 7) machine claim is directed to patentable matter, whereas the method claim is not directly so. The rejection has clearly pointed out how the claim as a whole amounts to a server that comprises merely objects, a browser and network interface, all of which mere software entities. Thus, the claimed subject matter is deemed not belonging to any of the four statutory categories; nor can the claim as a whole clearly establish the presence of hardware to carry out the software entities found in the claim as a whole in order to yield real world application results. The rejection is maintained because the argument is not persuasive.

35 USC § 103 Rejection:

(B) Applicants have submitted a disagreement that, as far as the Office Action rationale for combining teachings using Conallen and Chung, Chung is not appropriate for the teachings, because according to Chung, server side development uses JSP, client side uses JavaScript for scripting and XML for output (Appl. Rmrks pg. 8, bottom; pg 9, top). It is true that client browser uses JS script and XML for displaying or for carrying neutral formatting of other type of data; but that is well-known in the art. The rationale for rendering code conversion into target (JavaScript) JS constructs as effectuated in the Office Action flows right out of well-known concepts and existing proven-and-successful methodologies or prior methods; code for use in client browser like JavaScript. As it is evident, Chung teaches that JavaScript component can be

modeled from a class (see Table 4, pg. 6), which falls under the use of UML by Conallen to translate into script. Since Chung does not explicitly enforce (emphasis added) that developing code or translation into target code should not include the scenario wherein JavaScript code is target code, there is no reason to believe because of Chung's mention of JSP for development at server side -- which is also a well-known concept, not an invention by Chung - that target code for use at the client side cannot be in JS form construct. Rationale for obviousness is based on evidence from the references, well-known concepts or past proven methodologies, and inherent teaching as well as knowledge of one of ordinary skill in the art when faced with all of the above. The rationale has shown analogous endeavor in Chung and Conallen, a intended purposes, some known methodology, and reasons and motivation as to why one would be enclined to combine UML and code verification to effectuate script in Conallen such that the script when converted from the artifact of said UML would be in a client useable form like JavaScript construct, as well as explanation as to why by so doing a result would be beneficial. The above argument is deemed insufficient to overcome the Examiner's rationale as to why the combination would be appropriate.

(C) Applicants have submitted that 'mirror the object hierarchies and maintain the rules' is not found in Conallen and Chung (Appl. Rmrks pg. 9, 3rd para). The claim language is marred with improprieties and lack of support; as set forth above in the current Office Action. Broad interpretation has been used for addressing the merits of such deficient language. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the reference.

(D) Applicants have submitted that (Appl. Rmrks pg 9, middle) because JavaScript is proffered by Conallen as typeless, the claims have been amended to put forth the invention. The argument is deemed not commensurate with any claim language nor is it considered with the current grounds of rejection which is necessitated only by the amendments.

The claims will stand rejected as set forth above.

Conclusion

18. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.
Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Tuan A Vu/

Primary Examiner, Art Unit 2193

June 12, 2008